Лекция 12
Модели параллельного программирования

Ефимов Александр Владимирович
E-mail: alexandr.v.efimov@sibguti.ru

Курс «Архитектура вычислительных систем»
СибГУТИ, 2020

Оптимизация программного обеспечения

Цели:

- Минимизация времени выполнения
- Минимизация объема потребляемой памяти
- Минимизация энергопотребления

Способы:

- Алгоритмическая оптимизация кода
- Оптимизация доступа к памяти (кеш-памяти, ОЗУ, дисковой, сетевым хранилищам)
- Оптимизация ветвлений и загрузки конвейеров – Instruction Level Parallelism
- Векторизация кода – Data Parallelism (SSE, AVX, AltiVec, ARM SIMD)
- Распараллеливание – Thread Level Parallelism (OpenMP/Pthreads, CUDA/OpenCL/OpenACC, MPI)

Понятия параллельного программирования

- **Параллельный алгоритм**

  Описание процесса обработки информации, ориентированное на реализацию в коллективе вычислителей
- **Параллельная программа**

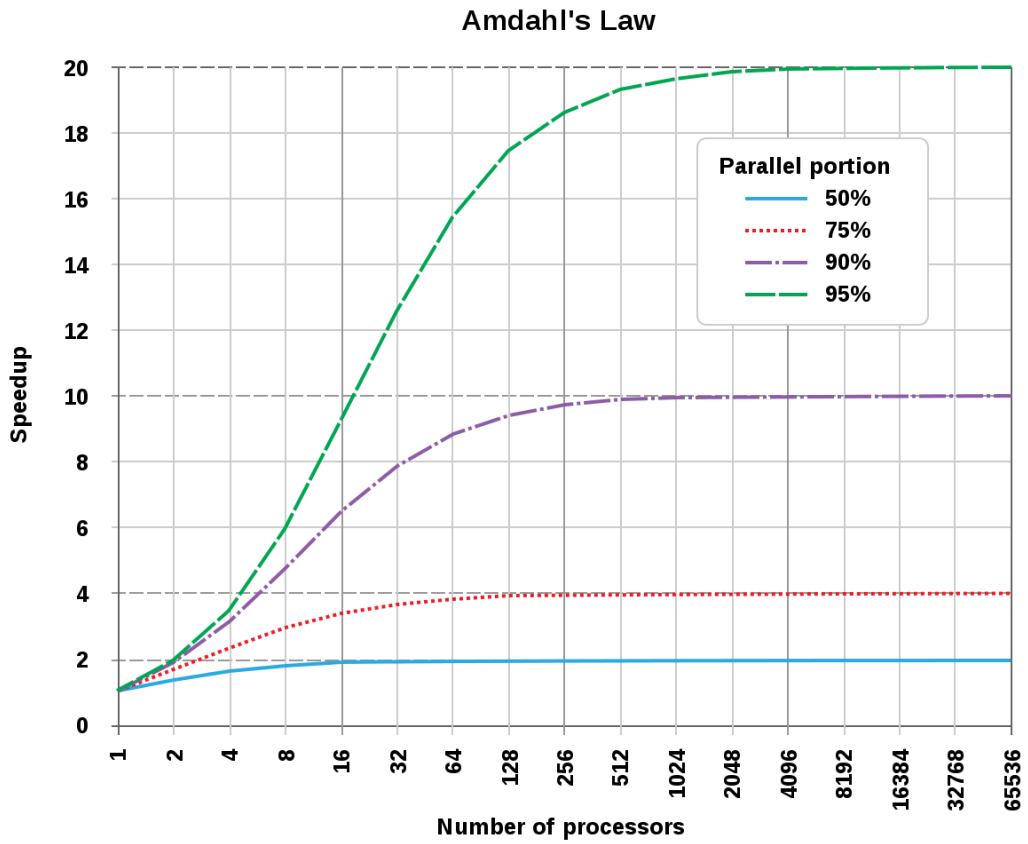
  Запись параллельного алгоритма на языке программирования, доступном коллективу вычислителей
- **Распараллеливание**

  Процесс «приспособления» методов к реализации на коллективе вычислителей или процесс «расщепления» последовательных алгоритмов решения сложных задач
- **Параллельное программирование**

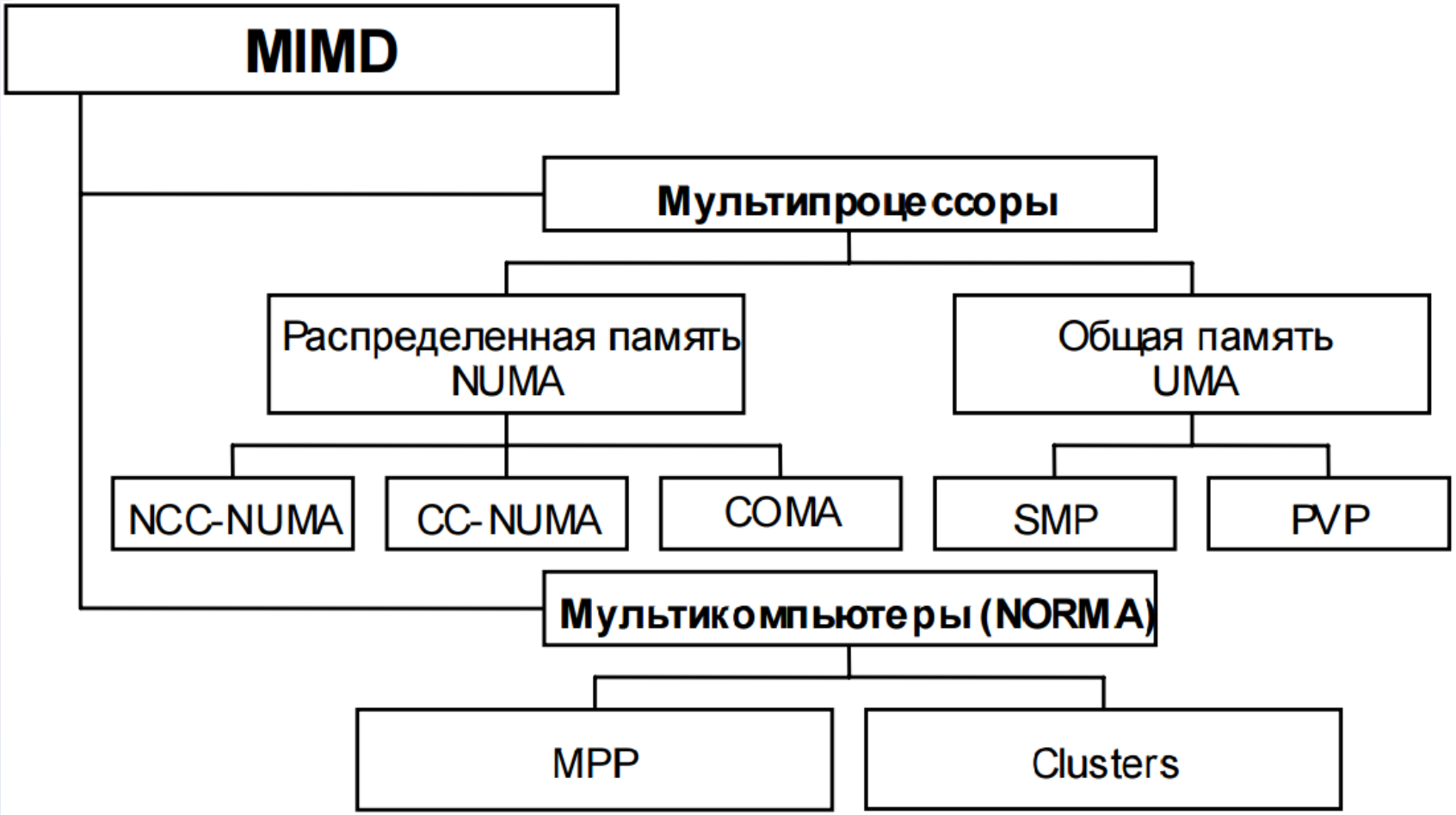
  Теоретическая и практическая деятельность по созданию параллельных алгоритмов и программ обработки информации

Закон Амдала

«В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого медленного фрагмента»



Классификация вычислительных систем

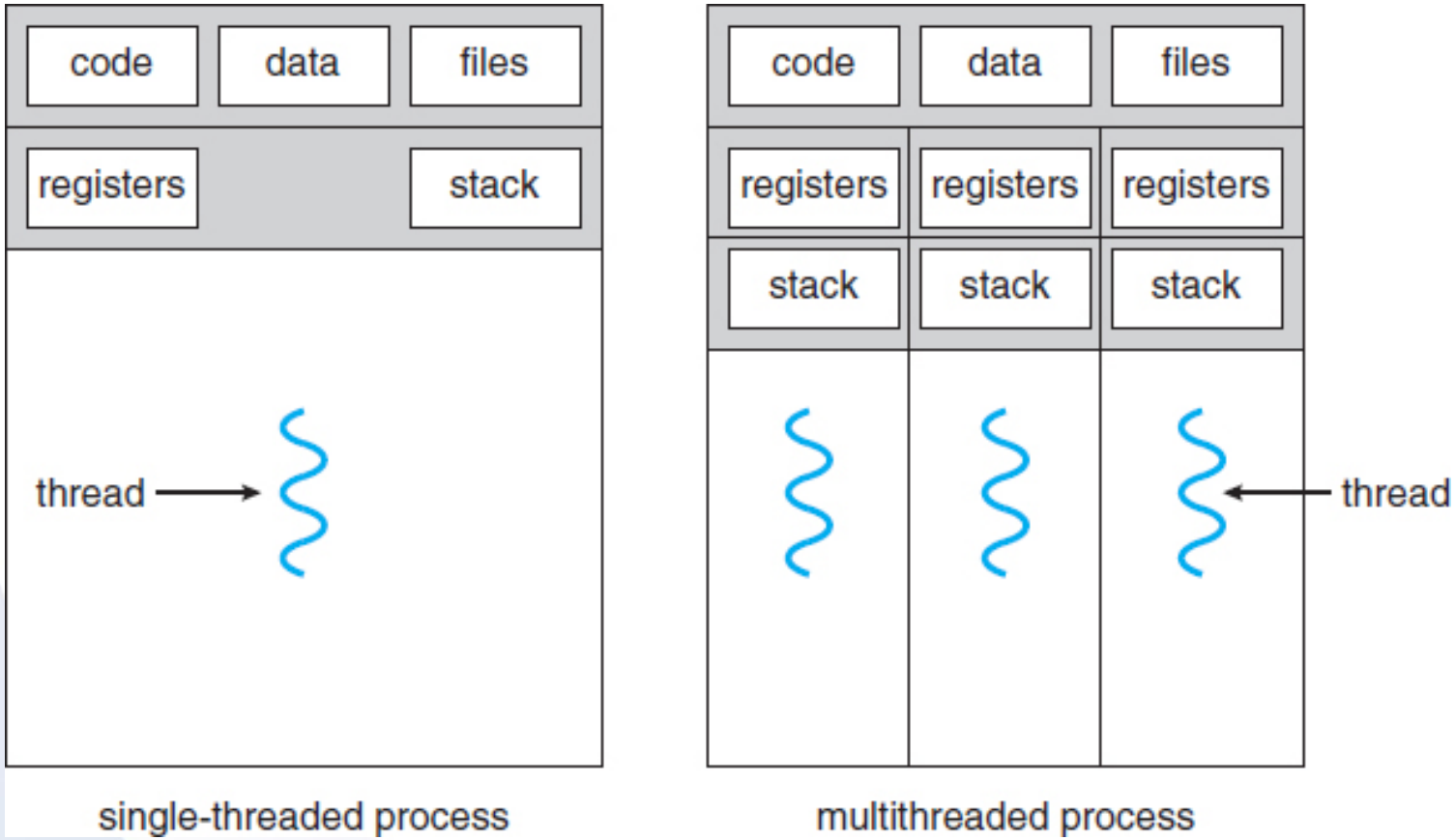


Стандарт POSIX

- **POSIX** (Portable Operating System Interface — переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов.
- **POSIX Threads** — стандарт POSIX-реализации потоков (нитей) выполнения.

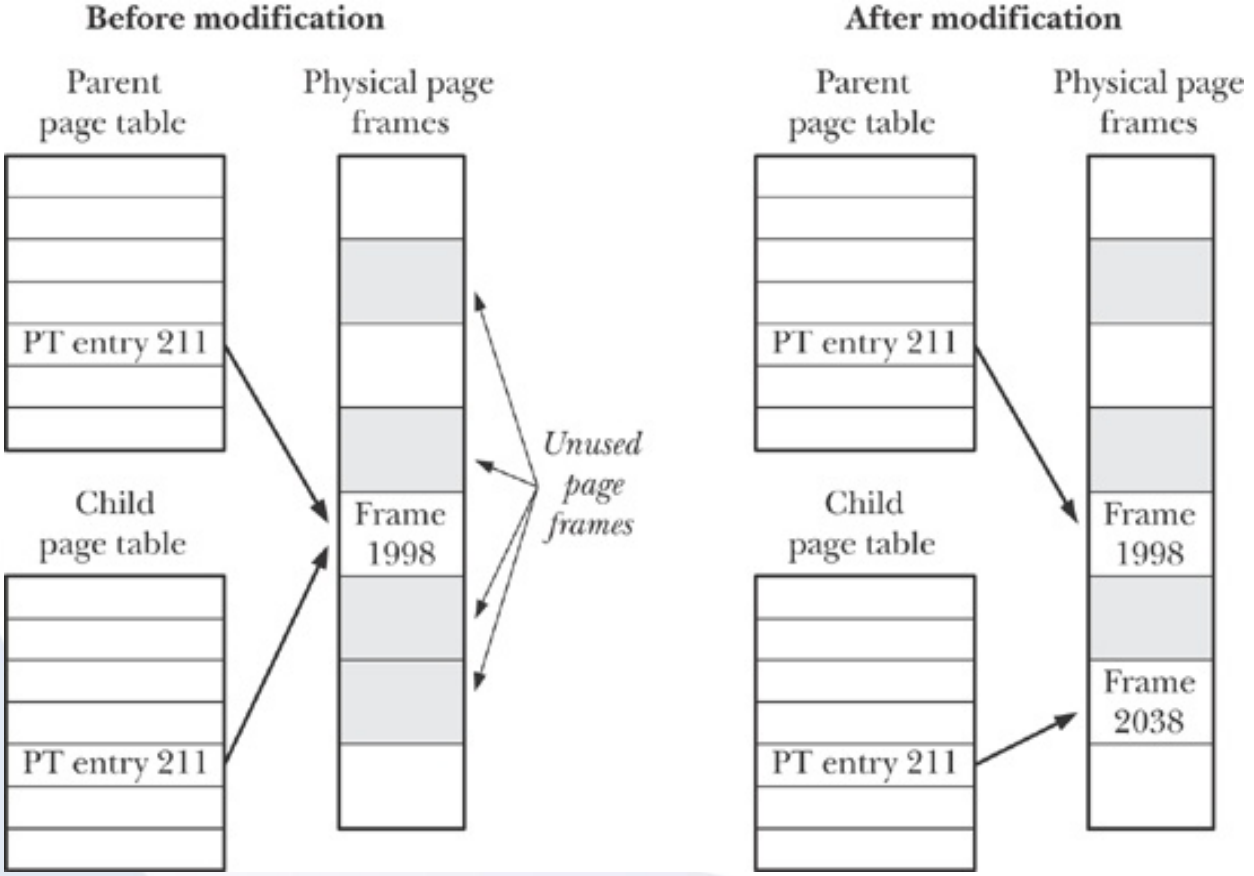
IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7. IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008) (Jan 2018), p. 1–3951.

Стандарт POSIX threads



Создание потока POSIX threads

```
#include <pthread.h>
int pthread_create (pthread_t *thread, const pthread_attr_t *attr,
                    void *(*start)(void *), void *arg);
```



Работа с потоками POSIX threads

- Завершение потока

void **pthread_exit**(void *retval);

int **pthread_cancel** (pthread_t THREAD_ID);

- Ожидание потока

int **pthread_join** (pthread_t THREAD_ID, void ** DATA);

- Синхронизация доступа к разделяемым ресурсам

int **pthread_mutex_init**(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

int **pthread_mutex_lock**(pthread_mutex_t *mutex);

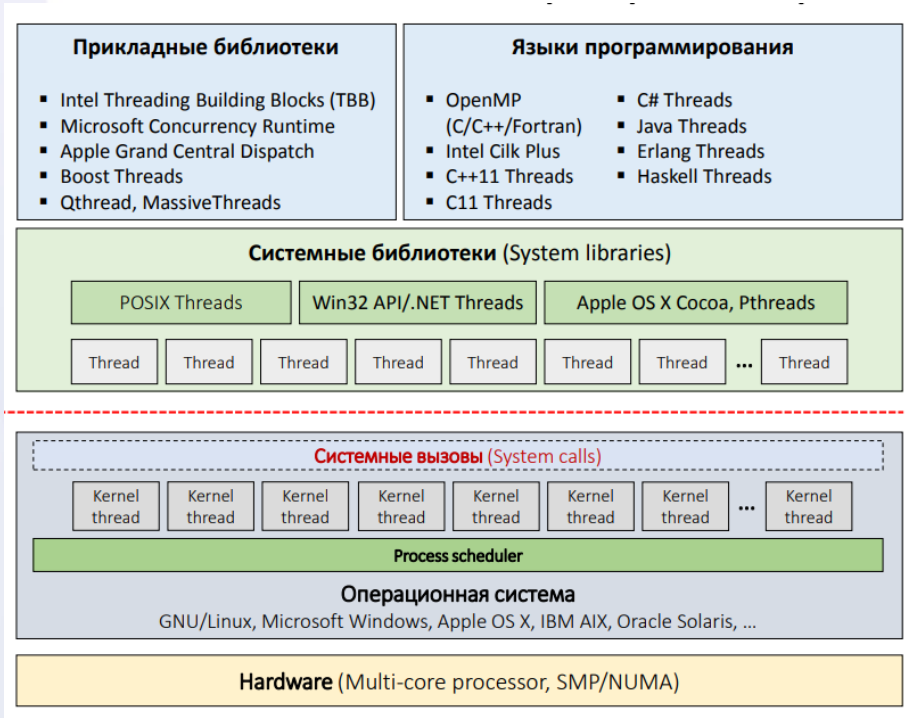
int **pthread_mutex_unlock**(pthread_mutex_t *mutex);

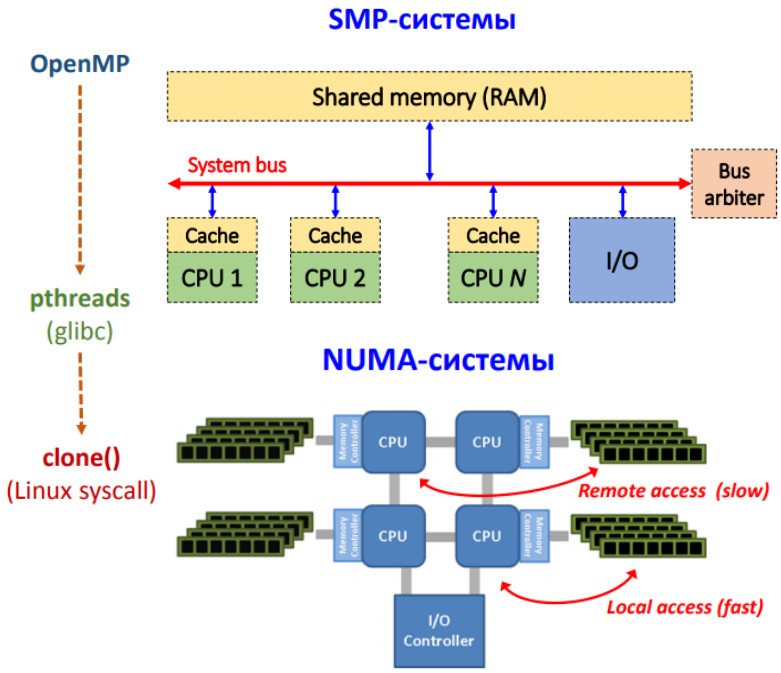
int **pthread_mutex_destroy**(pthread_mutex_t *mutex);

- Компиляция

gcc -o pthreads_test **-lpthread** pthreads_test.c

Параллельное программирование систем с общей памятью





http://www.mkurnosov.net/parprog/uploads/Main/isp-parprog2015-lec1.pdf

Стандарт OpenMP

- OpenMP (Open Multi-Processing) – стандарт, определяющий набор директив компилятора, библиотечных процедур и переменных среды окружения для создания многопоточных программ
- Разрабатывается в рамках OpenMP Architecture Review Board с 1997 года

OpenMP 2.5 (2005), OpenMP 3.0 (2008), OpenMP 3.1 (2011), OpenMP 4.0 (2013), OpenMP 4.5 (2015) OpenMP 5.0 (2018).

https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf

- Требуется поддержка со стороны компилятора

Разделяемое глобальное адресное пространство

- Языки программирования:

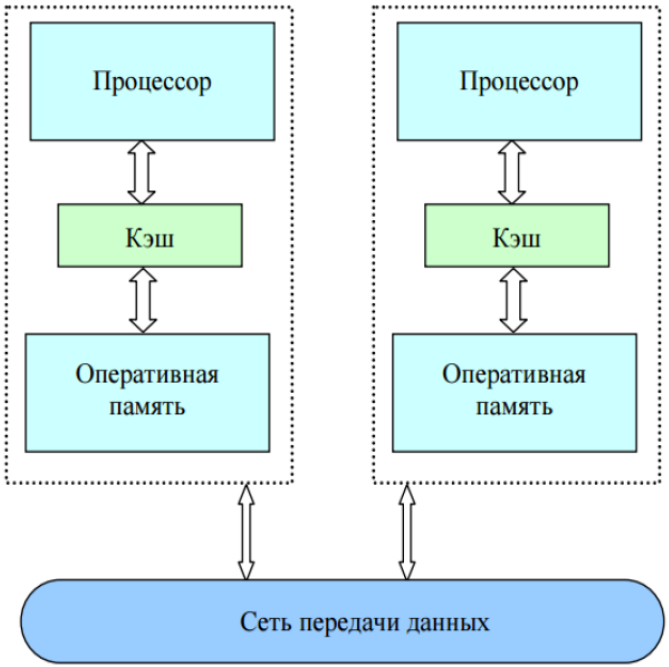
IBM X10

Cray Chapel

Параллельное программирование систем с распределенной памятью

Вычислительная система с распределенной памятью (distributed memory computer system) – совокупность вычислительных узлов, взаимодействие между которыми осуществляется через коммуникационную сеть (InfiniBand, Gigabit Ethernet, Cray Gemeni, Fujitsu Tofu, DragonFly, …)

- Каждый узел имеет множество процессоров/ядер, взаимодействующих через разделяемую память (shared memory);
- Процессоры могут быть многоядерными, ядра могут поддерживать одновременную многопоточность (SMT, Hyper-Threading).

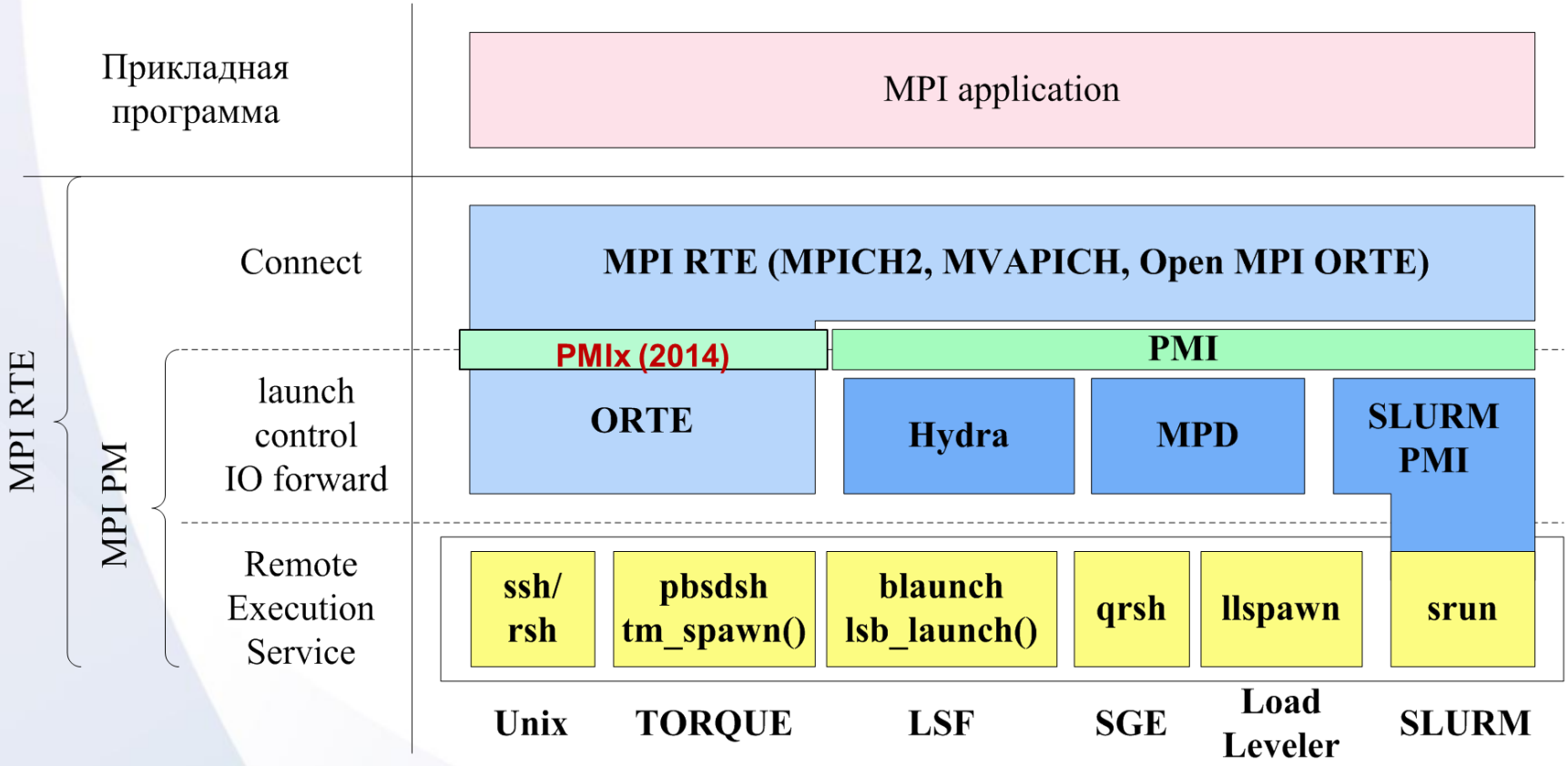


Создание процессов fork() и exec

- При выполнении вызова fork():

1) Выделяется память для описателя нового процесса в таблице процессов.
2) Назначается идентификатор процесса PID.
3) Создается логическая копия процесса, который выполняет fork() - полное копирование содержимого виртуальной памяти родительского процесса, копирование составляющих ядерного статического и динамического контекстов процесса-предка.
4) Увеличиваются счетчики открытия файлов (порожденный процесс наследует все открытые файлы родительского процесса).
5) Возвращается PID в точку возврата из системного вызова в родительском процессе и 0 - в процессе-потомке.

Стек технологий параллельного программирования



Стандарт MPI

- Message Passing Interface (MPI) – это стандарт на программный интерфейс коммуникационных библиотек для создания параллельных программ в модели передачи сообщений (message passing).
- Стандарт определяет интерфейс для языков программирования C и Fortran.
- Стандарт де-факто для систем с распределенной памятью (“ассемблер” в области параллельных вычислений на системах с распределенной памятью).
- Обеспечивает переносимость программ на уровне исходного кода между разными ВС (Cray, IBM, ...)

Реализации MPI

- MPICH (Open source, Argone NL, `http://www.mcs.anl.gov/research/projects/mpich3`)
  Производные от MPICH3: MVAPICH (MPICH for InfiniBand), IBM MPI, Cray MPI, Intel MPI, HP MPI, Microsoft MPI
- Open MPI (Open source, BSD License, http://www.open-mpi.org)
  Производные от Open MPI: Oracle MPI
- Высокоуровневые интерфейсы

C++: Boost.MPI, Java: Open MPI Java Interface, MPI Java, MPJ Express, ParJava, C#: MPI.NET, MS-MPI, Python: mpi4py, pyMPI

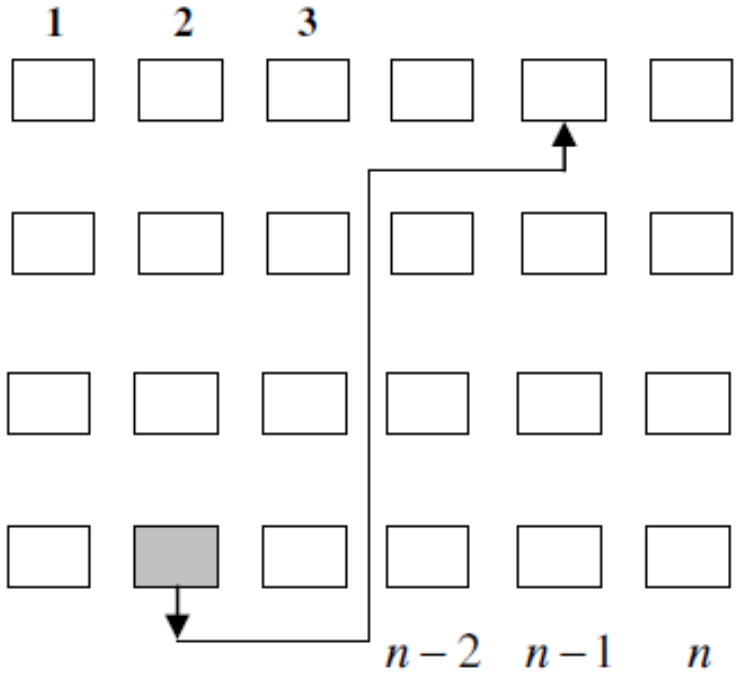
Отличия реализаций MPI

- Спектр поддерживаемых архитектур процессоров: Intel, IBM, ARM, Fujitsu, NVIDIA, AMD
- Типы поддерживаемых коммуникационных технологий/сетей: InfiniBand, 10 Gigabit Ethernet, Cray Gemeni, IBM PERCS/5D torus, Fujitsu Tofu, Myrinet, SCI, …
- Протоколы дифференцированных обменов двусторонних обменов (Point-to-point): хранение списка процессов, подтверждение передачи (ACK), буферизация сообщений, …
- Коллективные операции обменов информацией: коммуникационная сложность алгоритмов, учет структуры вычислительной системы (torus, fat tree, …), неблокирующие коллективные обмены (MPI 3.0, методы хранение collective schedule)
- Алгоритмы вложения графов программ в структуры вычислительных систем (MPI topology mapping)
- Возможность выполнения MPI-функций в многопоточной среде и поддержка ускорителей (GPU NVIDIA/AMD, Intel Xeon Phi)

Схемы обмена информацией между ветвями параллельных алгоритмов

- *Дифференцированный обмен (ДО, Point-to-Point)*

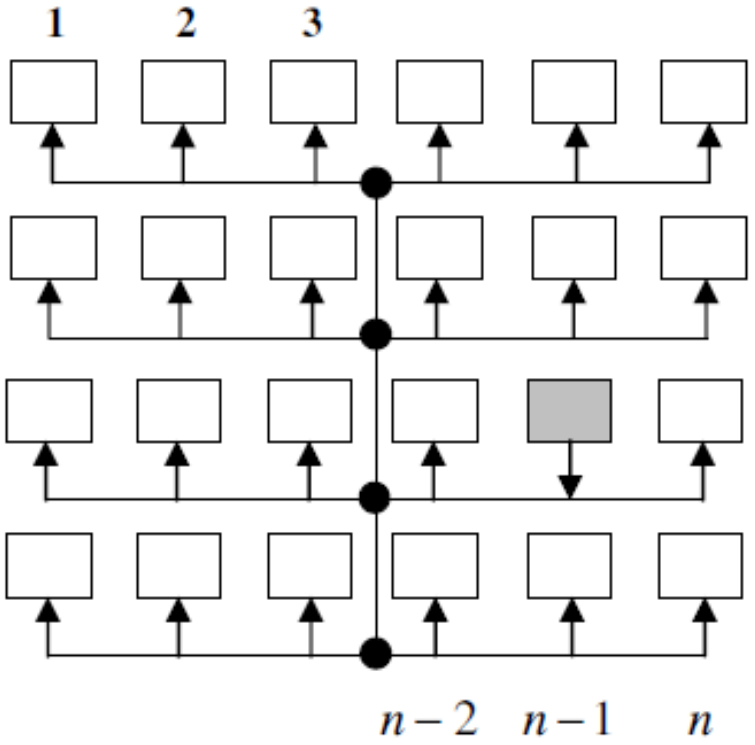
  Передача информации из одной ветви в любую другую ветвь или из одного вычислителя (передатчика) к другому вычислителю (приемнику)



Схемы обмена информацией между ветвями параллельных алгоритмов

- *Трансляционный обмен (TO, One-to-All Broadcast)*

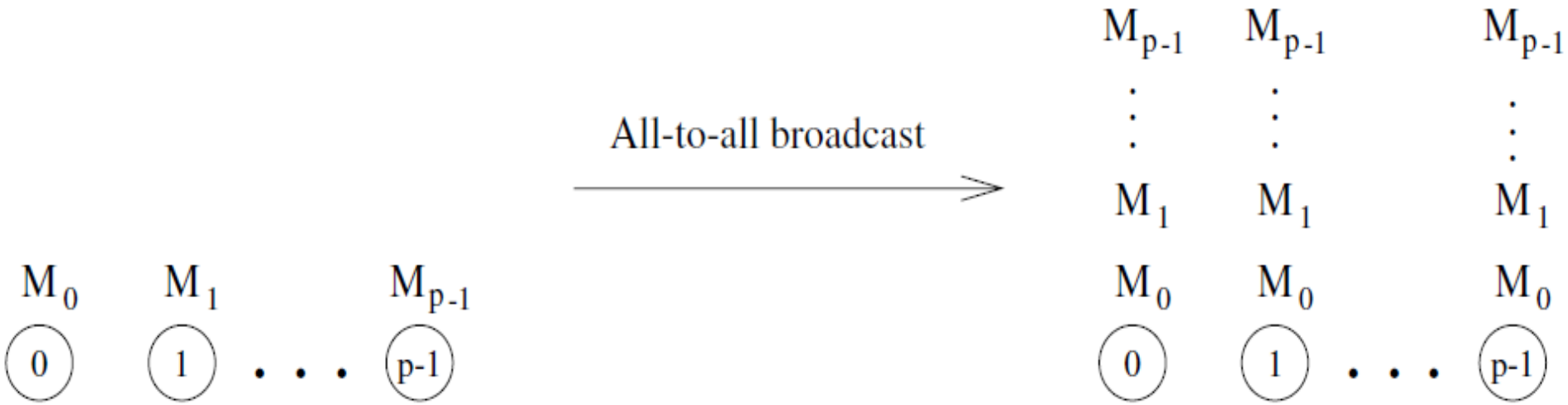
  Передача одной и той же информации из одной (любой) ветви одновременно во все остальные ветви параллельного алгоритма



Схемы обмена информацией между ветвями параллельных алгоритмов

- *Трансляционно-циклический обмен (ТЦО, All-to-all Broadcast)*

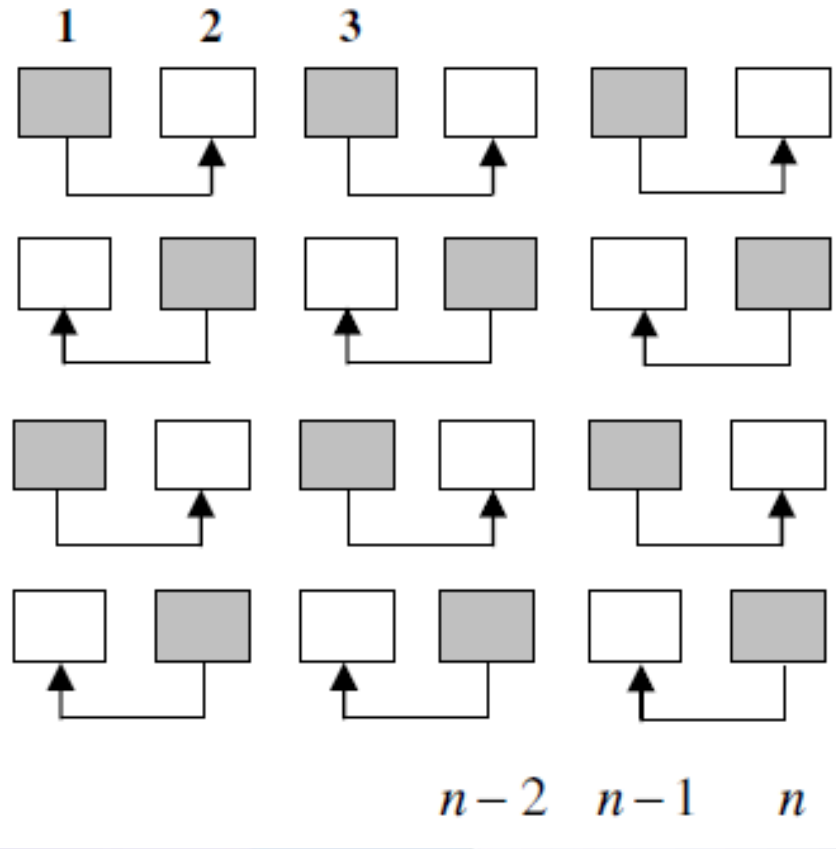
  Трансляция информации из каждой ветви во все

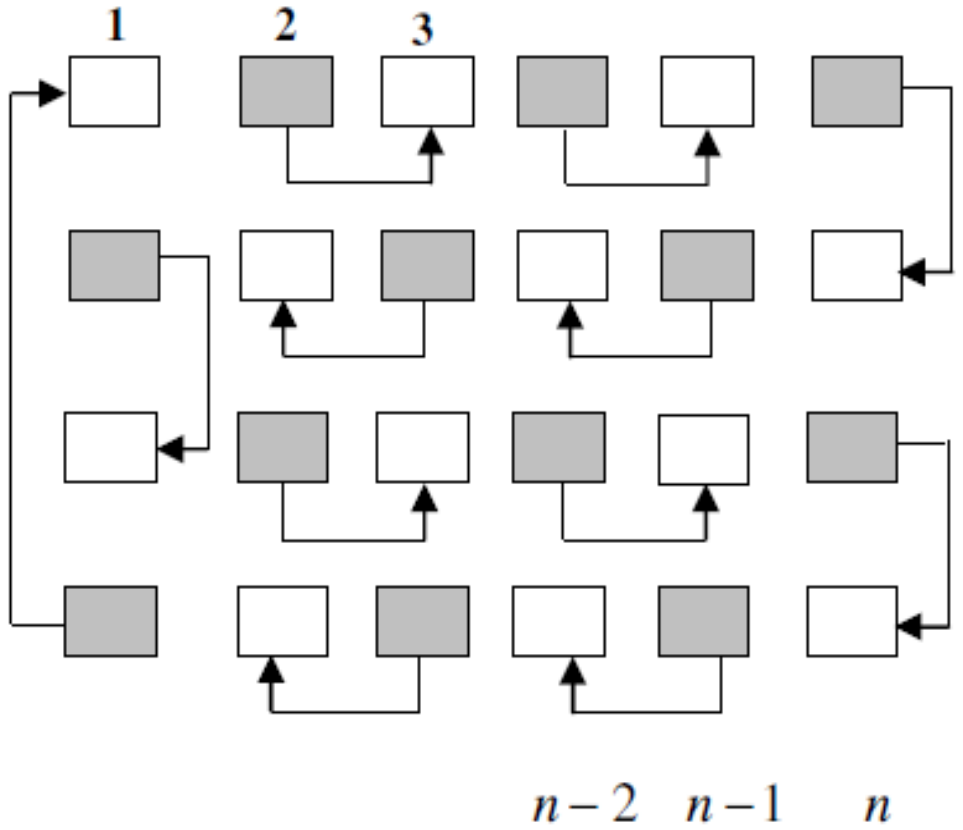


Схемы обмена информацией между ветвями параллельных алгоритмов

- *Конвейерно-параллельный обмен (КПО)*

  Передача информации между соседними ветвями

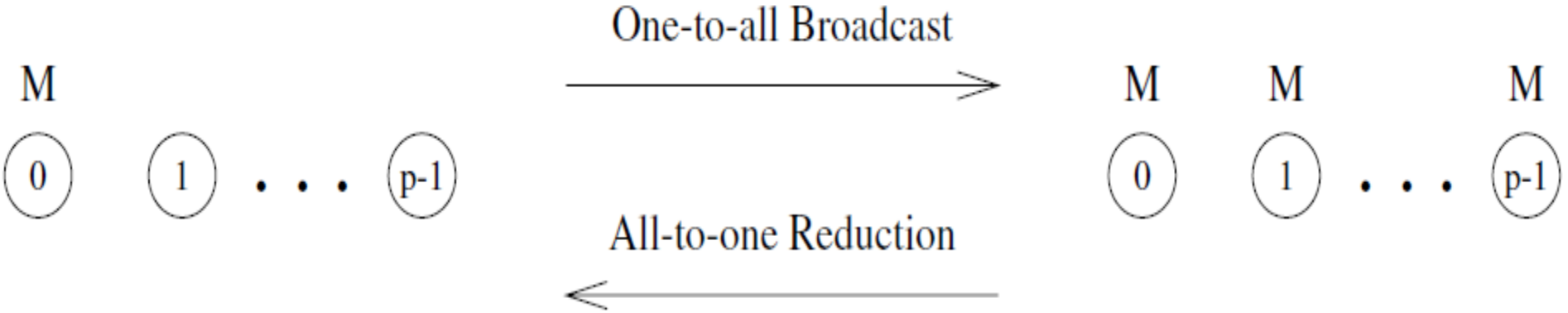




Схемы обмена информацией между ветвями параллельных алгоритмов

- *Коллекторный обмен (КО)*

  Инвертированный трансляционный обмен



Схемы обмена информацией между ветвями параллельных алгоритмов

Тип обмена	ДО	ТО	ТЦО	КПО	КО
Частота использования	2 %	17%	40%	34%	7%

Групповые схемы обмена информацией между ветвями параллельных алгоритмов или программ составляют более 90% от общего числа обменов

Методика распараллеливания сложных задач

- **Локальное распараллеливание**

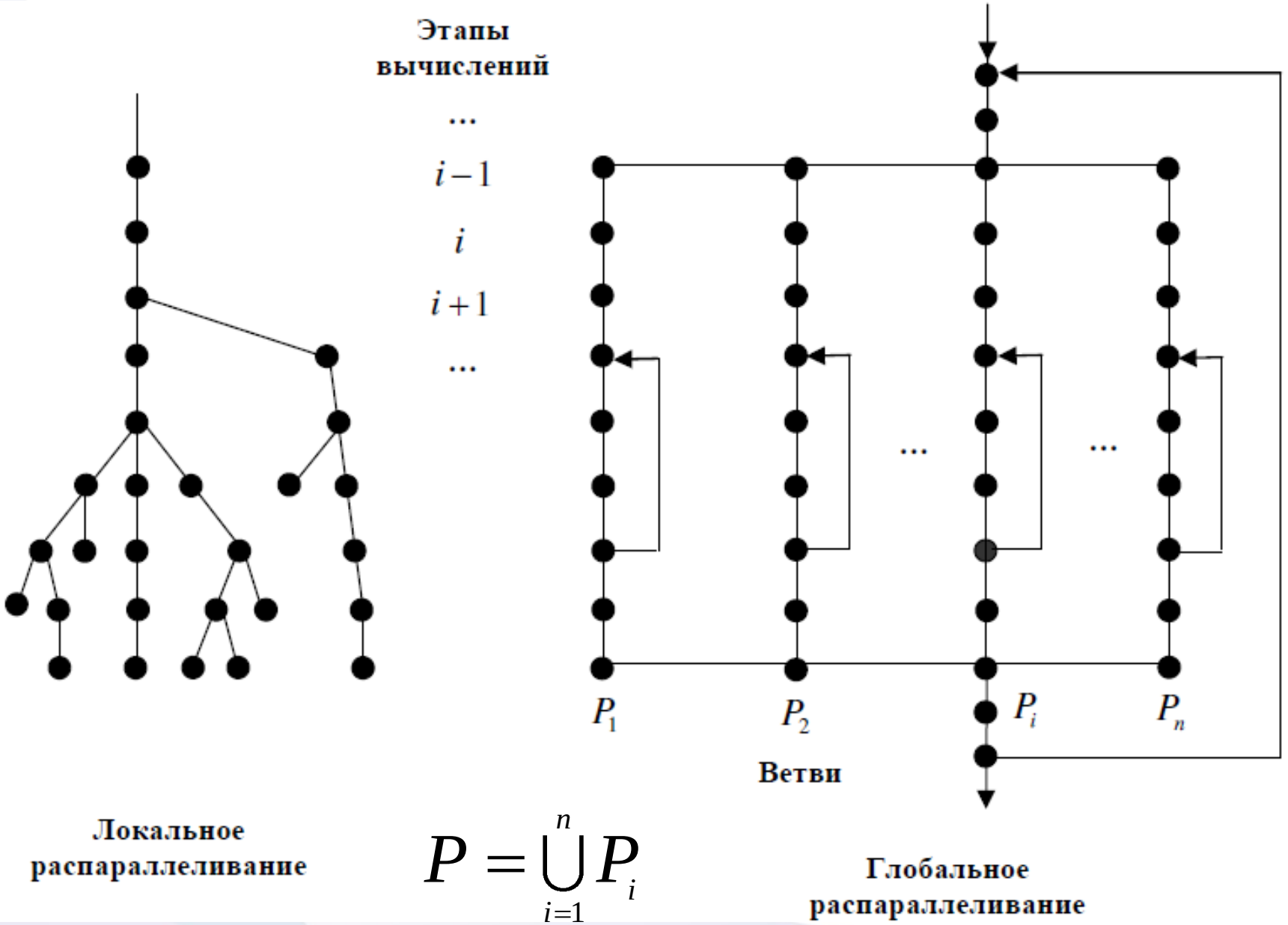
  Расщепление алгоритма решения сложной задачи на предельно простые блоки (операции или операторы) и выделение для каждого этапа вычислений максимально возможного количества одновременно выполняемых блоков

- **Глобальное (крупноблочное) распараллеливание**

  Разбиение сложной задачи на крупные блоки-подзадачи, между которыми существует слабая связность

  Такие подзадачи называют *ветвями параллельного алгоритма,* а соответствующие им программы – *ветвями параллельной программы*

Схемы параллельных алгоритмов



$$P = \bigcup_{i=1}^{n} P_i$$

Параллельный алгоритм умножения матриц

$$A[1:N,1:K]\times B[1:K,1:M]=C[1:N,1:M]$$

$$\begin{Vmatrix} a_{11} & a_{12} & \cdots & a_{1h} & \cdots & a_{1K} \\ a_{21} & a_{22} & \cdots & a_{2h} & \cdots & a_{2K} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i1} & a_{i2} & \cdots & a_{ih} & \cdots & a_{iK} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{N1} & a_{N2} & \cdots & a_{Nh} & \cdots & a_{NK} \end{Vmatrix} \times \begin{Vmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1M} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{h1} & b_{h2} & \cdots & b_{hj} & \cdots & b_{hM} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{K1} & b_{K2} & \cdots & b_{Kj} & \cdots & b_{KM} \end{Vmatrix} =$$

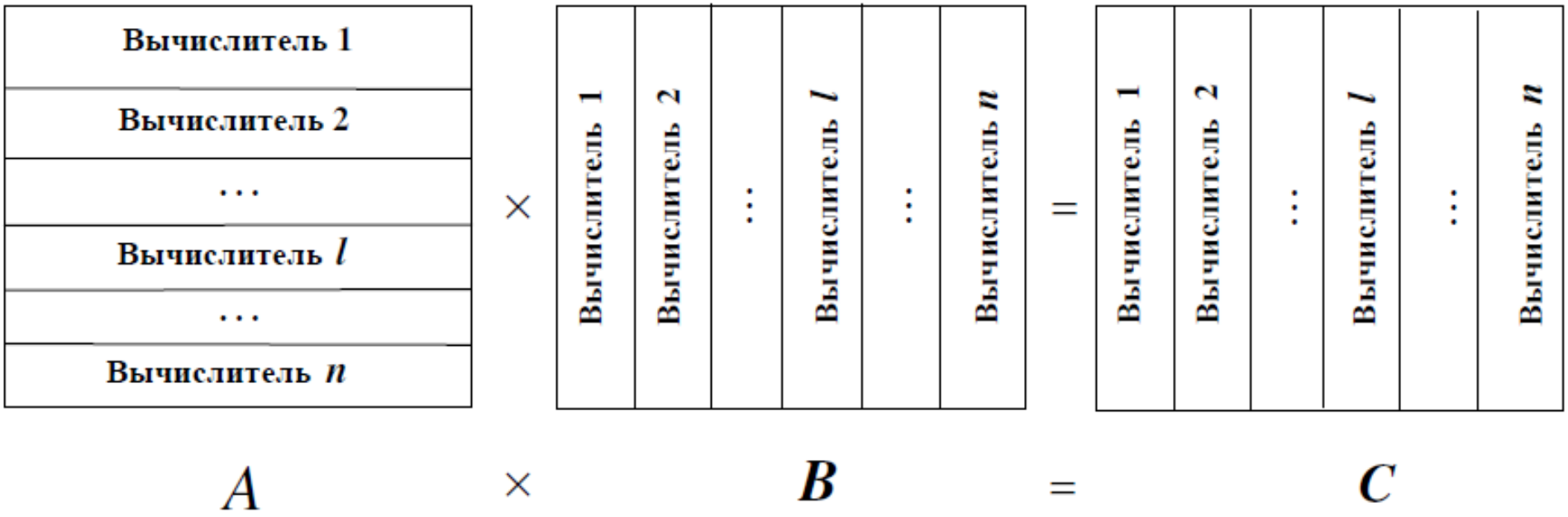
$$= \begin{Vmatrix} c_{11} & c_{12} & \cdots & c_{1j} & \cdots & c_{1M} \\ c_{21} & c_{22} & \cdots & c_{2j} & \cdots & c_{2M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{i1} & c_{i2} & \cdots & c_{ij} & \cdots & c_{iM} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{N1} & c_{N2} & \cdots & c_{Nj} & \cdots & c_{NM} \end{Vmatrix},$$

$$c_{ij}=\sum_{h=1}^{K} a_{ih}b_{hj}, \quad i=\overline{1,N}, \quad j=\overline{1,M}.$$

Последовательный алгоритм умножения матриц

```
1. procedure MAT_MULT (A, B, C)
2. begin
3.   for i:= 1 to N do
4.     for j:= 1 to M do
5.     begin
6.       C[i,j]:= 0;
7.       for h:= 1 to K do
8.         C[i,j]:= C[i,j] + A[i,h] * B[h,j];
9.     endfor;
10.end MAT_MULT
```

Параллельный алгоритм умножения матриц



Параллельный алгоритм умножения матриц

$N \times K = 4 \times 4$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix}$$

Вычислитель 1

Вычислитель 2

$K \times M = 4 \times 4$

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

Выч. 1 **Выч.2**

Количество вычислителей $n = 2$

Параллельный алгоритм умножения матриц

Вычислитель 1:

$$a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41} \quad a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}+a_{14}b_{42}$$

$$a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}+a_{24}b_{41} \quad a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42}$$

$$c_{11} \quad c_{12}$$

$$c_{21} \quad c_{22}$$

Параллельный алгоритм умножения матриц

Вычислитель 2:

$$a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} + a_{34}b_{43} \quad a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44}$$

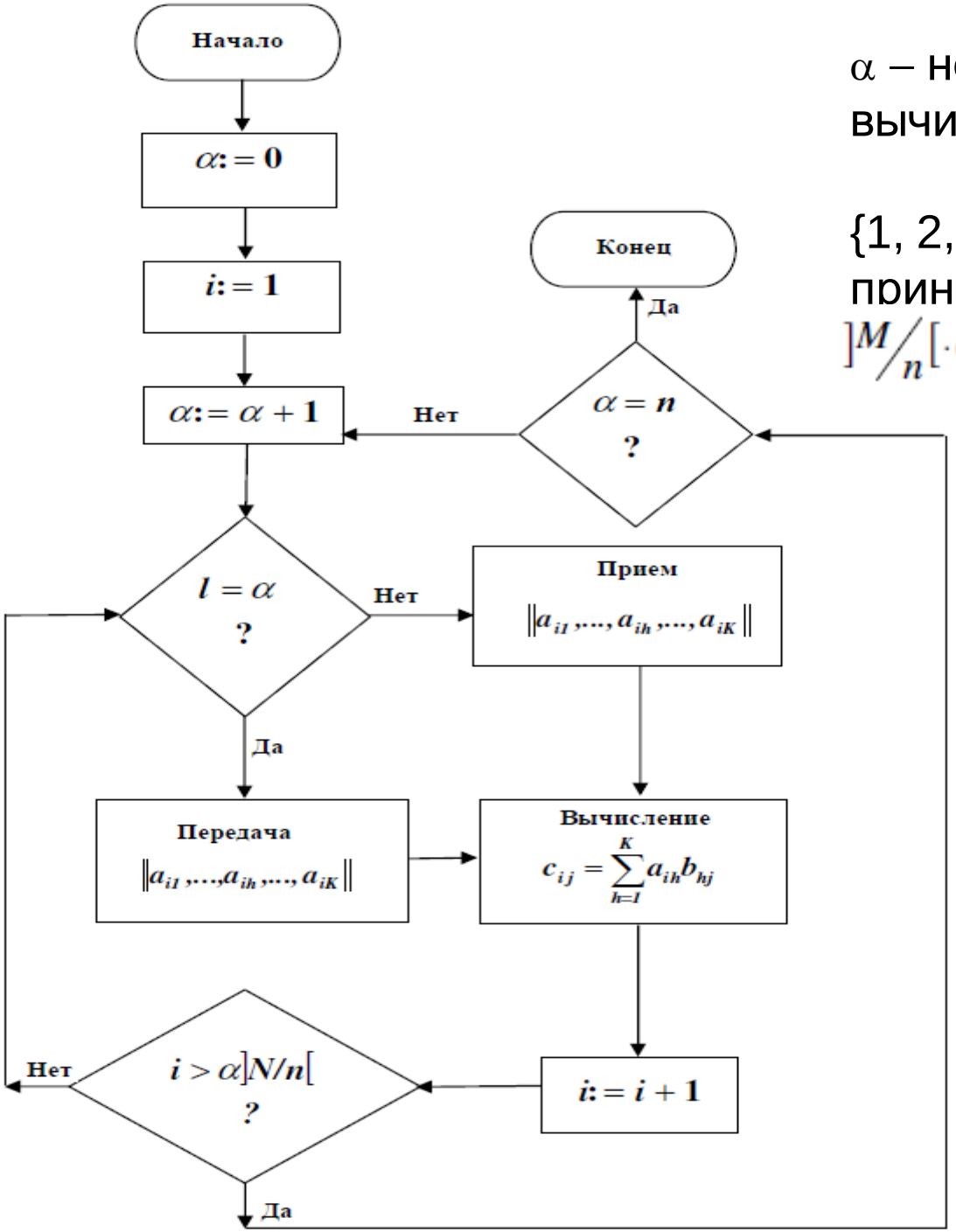
$$a_{41}b_{13} + a_{42}b_{23} + a_{43}b_{33} + a_{44}b_{43} \quad a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44}$$

$$c_{33} \quad c_{34}$$

$$c_{43} \quad c_{44}$$

Параллельный алгоритм умножения матриц

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} \quad ? \quad ? \quad \begin{pmatrix} c_{33} & c_{34} \\ c_{43} & c_{44} \end{pmatrix}$$



α – номер передающего вычислителя

{1, 2, …, α–1, α+1, …, n} – номера принимающих вычислителей

$$]M/n[\cdot(l-1) < j \leq]M/n[\cdot l$$

Показатели эффективности параллельных алгоритмов

- ✓ *Коэффициент накладных расходов*

$$\varepsilon = t / \mathrm{T},$$

t – время, расходуемое вычислительной системой на организацию и реализацию всех обменов информацией между ветвями,

T – время, требующееся на выполнение арифметических и логических операций при реализации алгоритма

- ✓ *Коэффициент ускорения*

$$\chi = \tau_1 / \tau_n ,$$

τ_1 – время выполнения алгоритма (решения задачи) на одном вычислителе,

τ_n – время выполнения соответствующего P -алгоритма в системе из *n*-вычислений

Показатели эффективности параллельных алгоритмов

- ✓ *Коэффициент ускорения параллельного алгоритма по сравнению с наилучшим последовательным алгоритмом*

$$\chi' = \tau'_1 / \tau_n,$$

τ'_1 – время реализации самого быстрого последовательного алгоритма на одном вычислителе.

- ✓ *Закон Амдала*

$$\chi^* \le \frac{\kappa}{\delta + (1-\delta)/n},$$

n – количество вычислителей, образующих ВС;

δ – относительная доля операций Р-программы, выполняемых последовательно;

κ – корректирующий коэффициент, $0 \le \kappa \le 1$.

Показатели эффективности параллельных алгоритмов

- ✓ *Коэффициент эффективности P-алгоритма*

$$E = \chi / n$$

- ✓ *Коэффициент эффективности P-алгоритма по отношению к наилучшему последовательному алгоритму*

$$E' = \chi' / n$$

Показатель эффективности для алгоритма умножения матриц

$$\varepsilon = \frac{t_n}{\rho(t_y + t_c)} = \frac{e}{\rho}, \qquad e = \frac{t_n}{t_y + t_c}$$

$t_п$ – время пересылки одного слова (элемента матрицы);

t_y и t_c – время выполнения операций умножения и сложения соответственно

$$\rho = \left]M/n\right[$$

$$\varepsilon \to 0 \text{ при } \rho \to \infty$$

Пример

Определить максимум коэффициента накладных расходов при реализации *p*-алгоритма на ВС, имеющей следующие параметры:

- разрядность *l* = 32;
- полосу пропускания канала между машинами *v* = 1 Гигабод;
- время выполнения операции сложения 0,1 мкс;
- время выполнения операции умножения 10 нс.

$$e = \frac{t_п}{t_с + t_у} = \frac{32 \cdot 10^{-9}}{0{,}1 \cdot 10^{-6} + 10 \cdot 10^{-9}} = \frac{0{,}000000032}{0{,}0000001 + 0{,}00000001} = 0{,}29$$

1 бод = 1 бит/с

1 Гигабод = 10^9 бод

Факторы, ограничивающие ускорение

- Время, расходуемое на синхронизацию параллельных ветвей и на обмены информацией между ними

- Несбалансированность (неоднородность) нагрузки вычислителей и (или) невозможность построения Р-алгоритма с числом ветвей, равным числу вычислителей в ВС

Парадокс параллелизма

- Достижение ускорения и эффективности параллельного алгоритма, превышающих значения *n* и 1, соответственно

$$\chi > n, \quad E > 1$$

«Парадокс параллелизма» - это более чем линейный рост производительности параллельной ВС с увеличением числа *n* её вычислителей

- Факторы, объясняющие этот парадокс
  - эффект памяти
  - возможность применения (априори параллельных) методов решения задач, реализация которых затруднительна на последовательной ЭВМ (например, снова из-за оперативной памяти, точнее ее ограниченного объема).

Понятие о сложных задачах

$$\varepsilon(V,n) = t(V,n)/\mathrm{T}(V,n)$$

V – количество операций, которые необходимо выполнить при решении задачи на ВС;
n – число параллельных ветвей или число вычислителей, на которых решается задача, $n \geq 2$;
t(V, n) – время, затрачиваемое на синхронизацию параллельных ветвей алгоритма, на настройку (программирование структуры) системы, на реализацию обменов информацией между ветвями (вычислителями);
T(*V, n*) – время, расходуемое системой на вычисления.

$$\varepsilon(V,n) \to 0 \quad \text{при} \quad V \to \infty$$

$$V \geq n \cdot 10^{k},$$

k – эмпирический коэффициент, $k \geq 1$.

$k \to 1$ при $\nu \to \nu^*$, где $1/\nu^*$ – время обращения к локальной памяти в вычислителе

Опыт применения методики крупноблочного распараллеливания сложных задач

1. Сложные задачи допускают построение параллельных алгоритмов, состоящих из идентичных ветвей.
2. При распределении исходных данных между ветвями параллельного алгоритма эффективным является принцип однородного расчленения массивов информации.
3. Для построения параллельных алгоритмов достаточно использовать пять схем обмена информацией между ветвями: дифференцированную, трансляционную, трансляционно-циклическую, конвейерно-параллельную и коллекторную схемы.
4. Простота организации обменов по рассмотренным схемам позволяет использовать в основном регулярные (однородные) структуры вычислительных систем.

Опыт применения методики крупноблочного распараллеливания сложных задач

5. Для записи параллельных алгоритмов решения сложных задач эффективны версии распространенных языков высокого уровня
6. Простота схем обмена и распределения данных по машинам ведет к простоте записи и реализации параллельных программ
7. Трансляционная, трансляционно-циклическая и конвейерно-параллельные схемы обмена информацией обеспечивают одновременную работу вычислителей системы и составляют не менее 90% всех схем, реализуемых в процессе выполнения параллельных программ сложных задач

Литература

Хорошевский В.Г. Архитектура вычислительных систем. Учебное пособие. – М.: МГТУ им. Н.Э. Баумана, 2005; 2-е издание, 2008.

Хорошевский В.Г. Инженерные анализ функционирования вычислительных машин и систем. – М.: “Радио и связь”, 1987.